TCP voorbeeld: gemeenschappelijk deel.

```java
public abstract class ChatBody extends Thread {

    private ObjectOutputStream output;
    private ObjectInputStream input;
    private String chatMessage;
    private final SimpleStringProperty displayArea = new SimpleStringProperty();
    private final SimpleBooleanProperty connected = new SimpleBooleanProperty();
    protected Socket socket;
    private final String name;
    private final String STOP;

    public ChatBody(String name, String stop) {
        this.name = name;
        STOP = stop;
    }

    @Override
    public abstract void run();

    public StringProperty displayAreaProperty() {
        return displayArea;
    }

    public BooleanProperty connectedProperty() {
        return connected;
    }

    public void updateDisplay(String mes) {
        displayArea.set(mes);
    }

    protected void getStreams() throws IOException {
        output = new ObjectOutputStream(socket.getOutputStream());
        output.flush();
        input = new ObjectInputStream(socket.getInputStream());
        updateDisplay("\nGot I/O streams\n");
    }

private void setConnected(boolean isConnected) {
    connected.set(isConnected);
}

public void sendMessage(String message) {
    try {
        output.writeObject(name + ">>> " + message);
        output.flush();
        updateDisplay("\n" + name + ">>> " + message);
    } catch (IOException ioException) {
        updateDisplay("\nError writing object");
    }
}
```

```
    protected void processConnection() throws IOException {
        String message = "Connection successful";
        sendMessage(message);
        setConnected(true);
        while (!message.equals(STOP)) {
            try {
                message = (String) input.readObject();
                updateDisplay("\n" + message);
            } catch (ClassNotFoundException ex) {
                updateDisplay("\nUnknown object type received");
            }
        }
    }

    protected void closeConnection() {
        updateDisplay("\nTerminating connection\n");
        setConnected(false);
        try {
            if (socket != null) {
                socket.close();
            }
        } catch (IOException ex) {
            updateDisplay(ex.getMessage());
        }
    }

}
```

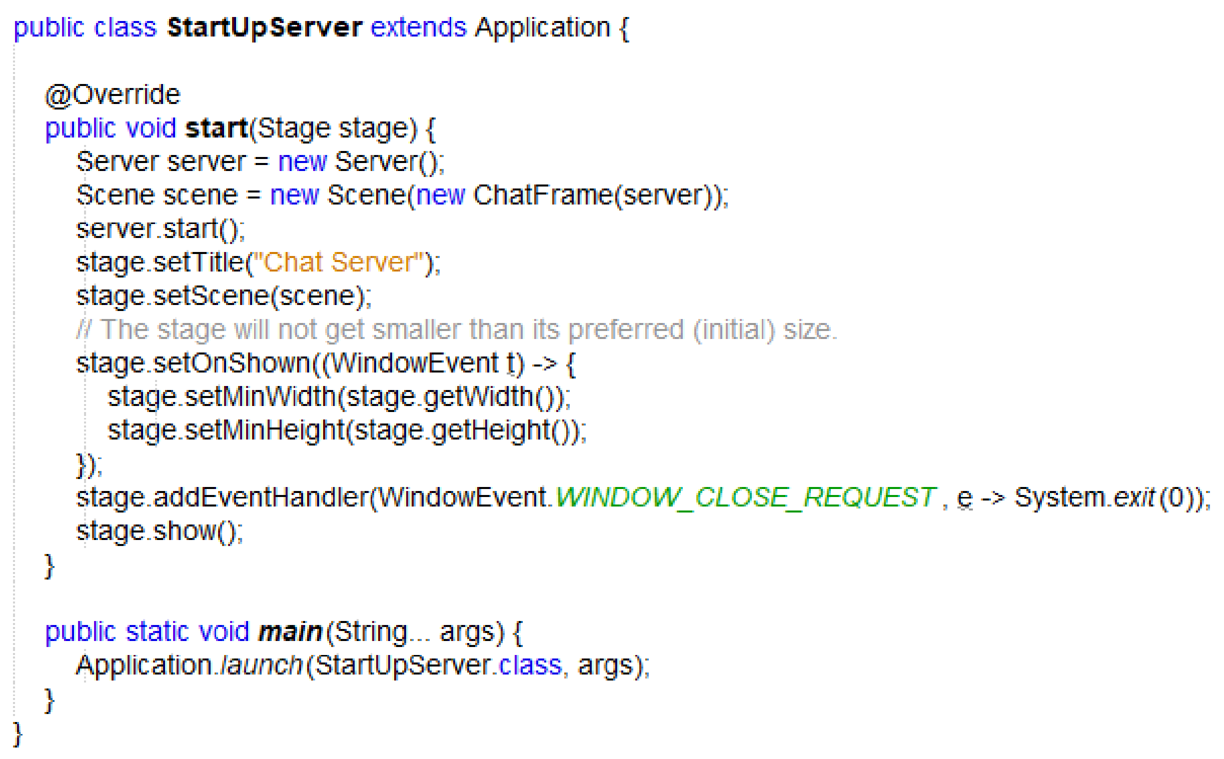
```
public class ChatFrame extends BorderPane {

    @FXML
    private TextField txtChatEntry;
    @FXML
    private TextArea txtChatDisplay;
    private final ChatBody chatBody;

    public ChatFrame(ChatBody chatBody) {
        this.chatBody = chatBody;
        FXMLLoader loader = new FXMLLoader(getClass().getResource("ChatFrame.fxml"));
        loader.setRoot(this);
        loader.setController(this);
        try {
            loader.load();
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
        chatBody.displayAreaProperty().addListener((observableValue, oldValue, newValue) ->
                Platform.runLater( () -> txtChatDisplay.appendText(newValue) )
        );
        chatBody.connectedProperty().addListener((observableValue, oldValue, newValue) ->
               Platform.runLater( () -> txtChatEntry.setEditable(newValue) )
        );
    }

    @FXML
    private void txtChatEntryAction(ActionEvent event) {
        chatBody.sendMessage(txtChatEntry.getText());
        txtChatEntry.clear();
    }
}
```

TCP voorbeeld: Server deel.



```
public class StartUpServer extends Application {

    @Override
    public void start(Stage stage) {
        Server server = new Server();
        Scene scene = new Scene(new ChatFrame(server));
        server.start();
        stage.setTitle("Chat Server");
        stage.setScene(scene);
        // The stage will not get smaller than its preferred (initial) size.
        stage.setOnShown((WindowEvent t) -> {
            stage.setMinWidth(stage.getWidth());
            stage.setMinHeight(stage.getHeight());
        });
        stage.addEventHandler(WindowEvent.WINDOW_CLOSE_REQUEST , e -> System.exit(0));
        stage.show();
    }

    public static void main(String... args) {
        Application.launch(StartUpServer.class, args);
    }
}
```

```
public class Server extends ChatBody {

    private ServerSocket server;
    private int numberOfConnections = 1;

    public Server() {
        super("SERVER", "CLIENT>>> TERMINATE");
    }

    @Override
    public void run() {
        runServer();
    }

    private void runServer() {
        try {
            server = new ServerSocket(12345, 100);
            while (true) {
                try {
                    makeConnection();
                    getStreams();
                    processConnection();
                } catch (EOFException eofException) {
                    updateDisplay("\nClient terminated connection");
                } catch (IOException ex) {
                    updateDisplay(ex.getMessage());
                } finally {
                    closeConnection();
                    ++numberOfConnections;
                }
            }
        } catch (IOException ex) {
            updateDisplay("\nNo ServerSocket : " + ex.getMessage());
        }
    }

    private void makeConnection() throws IOException {
        updateDisplay("Waiting for connection\n");
        socket = server.accept();
        updateDisplay("Connection " + numberOfConnections + " received from: "
                + socket.getInetAddress().getHostName());
    }

}
```

TCP voorbeeld: Client deel.

```
public class StartUpClient extends Application {

    @Override
    public void start(Stage stage) {
        String host = getParameters().getRaw().isEmpty()?"localhost":getParameters().getRaw().get(0);
        Client client = new Client(host);
        Scene scene = new Scene(new ChatFrame(client));
        client.start();
        stage.setTitle("Chat Client");
        stage.setScene(scene);
        // The stage will not get smaller than its preferred (initial) size.
        stage.setOnShown((WindowEvent t) -> {
            stage.setMinWidth(stage.getWidth());
            stage.setMinHeight(stage.getHeight());
        });
        stage.addEventHandler(WindowEvent.WINDOW_CLOSE_REQUEST , e -> System.exit(0));
        stage.show();
    }

    public static void main(String... args) {
        Application.launch(StartUpClient.class, args);
    }
}
```

```
public class Client extends ChatBody {

    private final String chatServer;

    public Client(String host) {
        super("CLIENT", "SERVER>>> TERMINATE");
        chatServer = host;
    }

    @Override
    public void run() {
        runClient();
    }

    private void runClient() {
        try {
            makeConnection();
            getStreams();
            processConnection();
        } catch (EOFException eofException) {
            updateDisplay("\nServer terminated connection");
        } catch (IOException ex) {
            updateDisplay(ex.getMessage());
        } finally {
            closeConnection();
        }
    }

private void makeConnection() throws IOException {
        socket = new Socket( InetAddress.getByName(chatServer), 12345 );
        updateDisplay("Connected to: " + socket.getInetAddress().getHostName());
    }

}
```

UDP voorbeeld: Server en Client.

```java
public class StartUp extends Application {

    @Override
    public void start(Stage stage) {
        String status = getParameters().getRaw().isEmpty()?"SERVER":getParameters().getRaw().get(0);
        String host= getParameters().getRaw().size()<2?"localhost":getParameters().getRaw().get(1);
        ChatBodyUDP chatUDPbody = new ChatBodyUDP(status, host);
        Scene scene = new Scene(new ChatFrame(chatUDPbody));
        chatUDPbody.start();
        stage.setTitle("Chat " + status);
        stage.setScene(scene);
        // The stage will not get smaller than its preferred (initial) size.
        stage.setOnShown((WindowEvent t) -> {
            stage.setMinWidth(stage.getWidth());
            stage.setMinHeight(stage.getHeight());
        });
        stage.addEventHandler(WindowEvent.WINDOW_CLOSE_REQUEST , e -> System.exit(0));
        stage.show();
    }

    public static void main(String... args) {
        Application.launch(StartUp.class, args);
    }
}
```

```
public class ChatBodyUDP extends Thread {

    private String chatMessage;
    private final StringProperty displayArea = new SimpleStringProperty();
    private final BooleanProperty connected = new SimpleBooleanProperty();
    protected DatagramSocket datagramSocket;
    protected DatagramPacket datagramPacket;
    protected int contactedPort;
    protected InetAddress contactedAddress;
    private final String name;
    private final String host;

    public ChatBodyUDP(String statusname, String hostName) {
        host = hostName;
        name = statusname;
    }

    @Override
    public void run() {
        try {
            if (name.equalsIgnoreCase("Server")) {
                datagramSocket = new DatagramSocket(5000);
            } else {
                datagramSocket = new DatagramSocket();
                contactedPort = 5000;
                try {
                    contactedAddress = InetAddress.getByName(host);
                } catch (UnknownHostException ex) {
                    updateDisplay(ex.getMessage());
                }
                setConnected(true);
                sendPacket("Client contacted");
            }
        } catch (SocketException ex) {
            updateDisplay("\nNo datagram socket : " + ex.getMessage());
        }
        waitForPackets();
    }
public StringProperty displayAreaProperty() {
    return displayArea;
}

public BooleanProperty connectedProperty() {
    return connected;
}

protected void updateDisplay(String mes) {
    displayArea.set(mes);
}
```

```
    private void setConnected(boolean isConnected) {
        connected.set(isConnected);
    }

    public void sendMessage(String message) {
        updateDisplay("\n" + name + ">>> " + message);
        sendPacket("\n" + name + ">>> " + message);
    }

    public void waitForPackets() {
        byte[] data = new byte[100];
        while (true) {
            try {
                DatagramPacket receivePacket = new DatagramPacket(data, data.length);

                datagramSocket.receive(receivePacket);
                contactedPort = receivePacket.getPort();
                contactedAddress = receivePacket.getAddress();
                updateDisplay("\nPacket received:"
                        + "\nFrom host: " + contactedAddress
                        + "\nHost port: " + contactedPort
                        + "\nLength: " + receivePacket.getLength()
                        + "\nContaining:\n\t" + new String(receivePacket.getData(),
                                0, receivePacket.getLength()));
                if (!connected.get()) {
                    setConnected(true);
                }
            } catch (IOException ex) {
                updateDisplay("\n " + ex.getMessage());
            }
        }
    }
   private void sendPacket(String message) {
       DatagramPacket sendPacket = new DatagramPacket(
               message.getBytes(), message.getBytes().length, contactedAddress, contactedPort);
       try {
           datagramSocket.send(sendPacket);
       } catch (IOException ex) {
           updateDisplay("Error send packet : " + ex.getMessage());
       }
   }

}
```

```java
public class ChatFrame extends BorderPane {

    @FXML
    private TextField txtChatEntry;
    @FXML
    private TextArea txtChatDisplay;
    private final ChatBodyUDP chatBody;

    public ChatFrame(ChatBodyUDP chatBody) {
        this.chatBody = chatBody;
        FXMLLoader loader = new FXMLLoader(getClass().getResource("ChatFrame.fxml"));
        loader.setRoot(this);
        loader.setController(this);
        try {
            loader.load();
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
        chatBody.displayAreaProperty().addListener((observableValue, oldValue, newValue) ->
                Platform.runLater( () ->  txtChatDisplay.appendText(newValue)) ) ;
        chatBody.connectedProperty().addListener((observableValue, oldValue, newValue) ->
               Platform.runLater( () ->   txtChatEntry.setEditable(newValue)) ) ;
    }

    @FXML
    private void txtChatEntryAction(ActionEvent event) {
        chatBody.sendMessage(txtChatEntry.getText());
        txtChatEntry.clear();
    }
}
```